Modern Scientific Software Management Using EasyBuild and Lmod

Markus Geimer (Jülich Supercomputing Centre, Germany)

Kenneth Hoste (HPC-UGent, Ghent University, Belgium)

Robert McLay (Texas Advanced Computing Center, TX, USA)

1st International Workshop on HPC User Support Tools (HUST-14)
in conjunction with SC'14, New Orleans, LA, USA
November 21, 2014

Motivation

HPC systems are typically used by large user communities.

- Widely varying demands
- Requires installation of many software packages
  - Sometimes identical/overlapping functionality (e.g., MPI libraries)
  - Multiple versions/builds

This is challenging for both users and administrators.

- *For users*: setting up the environment to use the desired software
  - Common solution: environment modules
- *For administrators*: installing software in a consistent way
  - Many HPC applications use non-standard build procedures
  - Successful installation steps often barely documented
  - Lack of sharing of good practices between HPC sites

Environment modules

- Shell-independent way to modify a user's environment
- Provide '`module`' command, evaluating output of a helper tool
  - Original implementations are Tcl-based (Tcl/C, Tcl-only)
  - Lmod, implemented in Lua
- Allows to, for example, list, load, unload, and swap modules
- Each module corresponds to a module file found in `$MODULEPATH`
  - Textual description of modifications to the user's environment (e.g., `$PATH`, `$CPATH`, `$LIBRARY_PATH`)
  - Additional specifications such as conflicts, help texts, etc.

Example: available modules, loading a module

```
$ module avail
foo/1.0   foo/1.7   bar/4.2
$ module load foo
$ module list
Currently Loaded Modulefiles:
 1) foo/1.7
```

Flat module naming schemes (I)

- HPC systems often feature multiple compilers & MPI libraries
  - Packages built with different compilers/MPIs should not be mixed
- Common solution: encode dependency in module name

Example: encoding compiler in name of MPI module

```
$ module avail OpenMPI
OpenMPI/1.7.3-GCC-4.8.2    OpenMPI/1.7.3-Intel-14.0
```

- Makes module names unwieldy for multiple dependencies

Example: long module names for application modules

```
$ module avail WRF
WRF/3.5-GCC-4.8.2-OpenMPI-1.7.3    WRF/3.5-Intel-14.0-MVAPICH2-1.9
```

- In many cases, packages additionally also depend on a set of mathematical libraries $\Rightarrow$ toolchains
  - Cryptic toolchain names (e.g., 'goolf')
  - Toolchain version w/o direct relationship to encapsulated packages

Flat module naming schemes (II)

- Total number of modules easily $\mathcal{O}(100)$
- Categorization can improve clarity

Example: categorization via module search paths

```
$ module avail
---------- <prefix>/compiler ----------
GCC/4.8.2   Intel/14.0   Clang/3.4
------------ <prefix>/mpi ------------
OpenMPI/1.7.3-GCC-4.8.2   OpenMPI/1.7.3-Intel-14.0
```

- Yet module listing can still be overwhelming
- Cumbersome to prevent loading incompatible modules
  - Names of conflicting modules must be explicitly listed in module files
  - Maintenance nightmare when adding/removing new packages conflicting with others

Installing scientific software

- Manual installation
  - Heavily relies on manpower of (part of) user support staff
  - Hard to enforce sharing of installation notes
- Package managers (rpm, yum, apt-get, etc.)
  - Limited support for installing multiple builds/versions of a package
  - Not well suited to idiosyncrasies of scientific software
- Scripting
  - Loosely coupled collection of scripts to automate installations
  - Often understood by only a few/single staff member(s)
  - Even when publicly released, rarely flexible enough to accommodate other sites needs

Wake-up call!

Although many HPC sites around the world face these problems, there is hardly any collaboration to address them!

Hierarchical module naming scheme

Key idea: make modules available step-by-step.

- Initially, only 'core' modules (e.g., compilers) are visible
- These extend `$MODULEPATH` on load, makes more modules available

Example: a simple module hierarchy

```
$ module avail
------------ <prefix>/Core ------------
GCC/4.8.2   Intel/14.0   Clang/3.4
$ module load GCC/4.8.2
$ module avail
------------ <prefix>/Core ------------
GCC/4.8.2   Intel/14.0   Clang/3.4
----- <prefix>/Compiler/GCC/4.8.2 -----
OpenMPI/1.7.3
```

Major advantages:

- Intuitive, short module names
- Only shows modules which are compatible in the current context

Challenges with using a module hierarchy

- Visibility of modules
  - Initially only core modules show up in output of '`module avail`'
  - How to locate packages w/o manually exploring the entire hierarchy?
- Awareness of changes to `$MODULEPATH`
  - Does swapping modules require reloading of other modules due to changes in the module search path?
- Module availability on different paths in the hierarchy
  - What if a dependent module is not available in the target module search path after swapping a module lower in the hierarchy?
- Creating and maintaining a module hierarchy requires special care
  - Appropriate hierarchy level, correctly handling dependencies, ...

Software build and installation tools

- SWTools (NICS/ORNL)
  - Only one public release (2011)
- Smithy (NICS/ORNL)
  - Follow-up to SWTools, also supports formulas (∼80 packages)
  - Available on GitHub, since 2013 mostly bug fixes
- iBS (iVEC)
  - Not yet publicly available
- Spack (LLNL)
  - Powerful and well-documented command-line interface
  - Supports ∼50 packages
  - Available on GitHub

Houston, we *had* a problem!

- No support for organizing modules hierarchically
- Lack of sizable communities up until now

Module tools

Different implementations of environment modules system:

- Cmod (C based): now obsolete (last updated in 1998)
- Tcl-based implementations (Tcl/C or Tcl-only): common practice
- pymodule (Python based): experimental/incomplete

Tools similar to environment modules (no longer actively developed):

- Dotkit: last updated in 2008
- Softenv: last updated in 2007

Houston, we *had* a problem!

Module hierarchy usability challenges are not addressed by any of these.

Tools for dealing with a module hierarchy

Adequate tools for dealing with a module hierarchy are indispensable.

- Modules tool that users interact with must be *hierarchy-aware*
- Complexity of maintaining a module hierarchy begs for *automation*

Two recent tools provide the necessary support:

- **Lmod**: `https://www.tacc.utexas.edu/tacc-projects/lmod`
  - Specifically developed for using a hierarchical module tree while supporting flat module layouts
  - Provides required hierarchy-specific features
- **EasyBuild**: `http://hpcugent.github.io/easybuild`
  - Automates software installation process, generates module files
  - Provides full control over module naming scheme
  - Includes support for organizing modules hierarchically

Lmod: a modern modules tool

- Drop-in alternative for Tcl-based module tools (a few edge cases)
- Improves user experience, without hindering experts
- Written in Lua, available since Oct'08, reads Tcl module files
- Frequent releases, driven by community demands and feedback

Example: swapping modules using `ml` shorthand in a module hierarchy

```
$ ml
Currently loaded modules:
 1) GCC/4.8.2   2) MPICH/3.1.1   3) FFTW/3.3.2
$ ml -GCC Clang
The following have been reloaded:
 1) FFTW/3.3.2   2) MPICH/3.1.1
$ ml
Currently loaded modules:
 1) Clang/3.4   2) MPICH/3.1.1   3) FFTW/3.3.2
```

Lmod: feature highlights

- *Module hierarchy-aware* design and functionality
  - Searching across entire module tree with '`spider`' subcommand
  - Automatic reloading of dependent modules on '`module swap`'
  - Marking missing dependent modules as inactive after '`module swap`'
- Caching of module files, for responsive subcommands (e.g., `avail`)
- Site-customizable behavior via provided hooks
- `ml` shorthand command, load/unload shortcuts
- Various other useful/advanced features, including:
  - Case-insensitive '`avail`' subcommand
  - Can send subcommand output to stdout (rather than to stderr)
  - Defining module families (e.g., '`compiler`', '`mpi`')
  - Assigning properties to modules (e.g., '`Phi-aware`')
  - Stack-based definition of environment variables (`pushenv`)
  - User-definable collections of modules

EasyBuild: building software with ease

- Open source (GPLv2) framework for building and installing software
- Collection of Python packages and modules
- Original implementation by HPC-UGent, since 2009
- Thriving community: actively contributing, driving development
- New release every 4–6 weeks
  - Latest release: EasyBuild v1.15.2 (Oct'14)
- Supports over 500 different software packages
  - Including CP2K, NAMD, NWChem, OpenFOAM, PETSc, QuantumESPRESSO, WRF, ...
- Well documented: `http://easybuild.readthedocs.org`

EasyBuild: feature highlights

- Fully autonomously building and installing (scientific) software
  - Automatic dependency resolution
  - Automatic generation of module files
- Thorough logging of executed procedure
- Highly configurable, via config files/environment/command line
- Dynamically extendable with additional easyblocks, toolchains, etc.
- *Support for module hierarchies*, via custom module naming scheme

Example: building/installing WRF & dependencies *with a single command* (!)

```
$ eb WRF-3.5-goolf-1.4.10.eb --robot
$ module spider WRF
...
This module can only be loaded through the following modules:
GCC/4.7.2, OpenMPI/1.6.4
```

EasyBuild: high-level design overview

- EasyBuild *framework*
  - Core of EasyBuild
  - Provides supporting functionality for building and installing software
- *easyblock*
  - Python module
  - Implements a (generic) software build/install procedure
- *easyconfig* file
  - Build specification: software name/version, toolchain, etc.
- Compiler *toolchain*
  - Compilers with accompanying libraries (MPI, BLAS/LAPACK, etc.)

Putting it all together

The EasyBuild *framework* leverages *easyblocks* to automatically build and install (scientific) software using a particular *compiler toolchain*, as specified by one or multiple *easyconfig files*.

Example use case

Build and install WRF in a module hierarchy

```
$ export EASYBUILD_MODULE_NAMING_SCHEME=MyHMNS
$ eb WRF-3.5-goolf-1.4.10.eb --robot
$ eb WRF-3.5-ictce-5.3.0.eb --robot
```

List existing WRF modules, load GCC build

```
$ module spider WRF
...
$ ml GCC OpenMPI WRF
```

Swap to Intel build of WRF

```
$ ml -GCC -OpenMPI +icc +ifort +impi
The following have been reloaded:
...
```

EasyBuild and Lmod communities

- Both EasyBuild and Lmod have vibrant communities
- Estimating their sizes is difficult, though
  - EasyBuild
    - Over 80 subscribers to mailing list
    - About a dozen active members on `#easybuild` IRC channel
    - Users and contributors at HPC sites around the world (e.g., JSC, Stanford, U. Auckland, Bayer AG, ...)
    - Six 3-day EasyBuild 'hackathons', at various European HPC sites
  - Lmod
    - $\sim$50 subscribers to mailing list
    - Deployed by a couple of hundred HPC sites (e.g., Stanford, Harvard, TACC, U. Warwick, JSC, Total, NASA, ...)
    - Number of users $\mathcal{O}(10{,}000)$
- Many sites/users contribute by
  - Requests, suggestions, and bug reports
  - Sharing patches and implementing new features

Synergy between EasyBuild and Lmod

- EasyBuild can easily build and install hundreds of packages
  - ⇒ Lots of modules, overwhelming for users
- Lmod's support for hierarchical modules trees can help
  - ⇒ Support for using Lmod and hierarchical module naming schemes was added to EasyBuild
- EasyBuild uncovered performance issues in Lmod
  - ⇒ Lmod has significantly improved the speed of certain operations (e.g., `module --terse avail`)
- Feature requests from the EasyBuild community
  - ⇒ Lmod has added new functionality, for example stack-based definition of environment variables using '`pushenv`'

Bottom line

Synergy between EasyBuild and Lmod has made both tools significantly better!

Future work

- Add support for Lmod-specific features in EasyBuild
  - Properties
  - Families
  - Non-strict version loads: `load(atleast("GCC","4.8"))`
- The hierarchy concept needs further investigation
  - Lmod currently supports a single hierarchy: Core ⇒ Compiler ⇒ MPI
  - Traditional EasyBuild toolchains also contain multiple math libraries ⇒ Multi-dimensional module hierarchy ("matrix")
- Improve EasyBuild's dependency resolution mechanism to support subtoolchains
- Extend EasyBuild to support multiple module naming schemes concurrently (e.g., to gradually move from one layout to another)

Conclusion

- Building software for HPC systems is difficult and painful
- Flat module layout is common practice, but has many drawbacks
  - Overwhelming number of modules
  - Unwieldy module names due to dependencies
  - Loading compatible modules is the user's responsibility
- Hierarchical module organization helps significantly
  - Fewer visible modules (only compatible ones), short module names
  - Requires adequate tool support, however
- EasyBuild
  - Automates building and installing (scientific) software, incl. modules
  - Supports maintaining hierarchical module trees
  - Helps to share knowledge between HPC centers
- Lmod
  - Provides required hierarchy-specific features (and a lot more)
- Both tools significantly benefit from collaboration and communities

Thank you!

Questions?